CycADS User Manual

1. **System overview** **2**

2. **The CycADS database** **2**

3. **The programs** **3**

3.1. Loader programs *3*

GBKLoader 3

GFF3Loader 5

KOLoader 7

SubseqDbxrefAnnotationLoader 7

EntityDbxrefAnnotationLoader 8

DbxrefDbxrefAnnotationLoader 9

EntitySynonymLoader 9

DbxrefSynonymLoader 9

3.2. Program to extract annotations to a file *10*

The command line 10

Config.properties 10

3.3. Auxiliary programs *12*

4. **Usage examples** **12**

5. **System requirements** **12**

6. **Installation** **13**

7. **Troubleshooting** **13**

8. **Future works** **13**

CycADS is a system designed and developed to build a BioCyc database using many different annotation sources. The system is on GNU License and can be downloaded freely (including the source code) from http://code.google.com/p/cycads/.

1. System overview

We made one very flexible system to import data from several different file formats into a database and to export data from the database to a file format (.pf file) specifically required by the Pathologic program of the PathwayTools system. This "pf file" can be generated with different filters and options. The filters can be applied to the external reference links and to the annotations used by PathwayTools (EC and GO numbers). The system CycADS was developed in Java and therefore can be executed in many different operating systems (see the section System requirements for details).
The system code can be split logically in 3 layers:
Data: Classes and interfaces to access and represent the entities used in the system. We can split this layer in 2 packages:
Database storage. The classes responsible to store and retrieve the data from/to the repository. Currently, we are storing the data in a SQL database system and we are using SQL queries and SQL commands in the classes of this layer.
Database access. Represents logically the data and is the interface to access this layer. With the existence of this package we can change the way the data is stored (database storage package) without interfering with the other layers. We are using Java interfaces to develop this layer.

Logical: This layer coordinates the commands requested, makes logical decisions and interacts with the data layer. This layer includes the information and logic to parse several different data format.
User interface: This layer makes the interaction with the users. It gets the data provided by the users, triggers the logical processes and presents the data generated by the system. Currently, the user passes information to CycADS (or arguments of the system) through the command line parameters and through the config.properties file.

2. The CycADS database

We have some basic entities stored in the CycADS database, which are:
Organism: an organism.
Sequence: a chromosome or contig of the organism for a given genome assembly (with information about the assembly version). The objects of this entity can also include the DNA sequence, but this is not mandatory.
Subsequence: reference to a sequence fragment defined by a set of positions on a sequence, indicating the beginning and the end of the fragment on the sequence, as well as the beginning and end of the intron(s) where it applies.
DBxRef: reference to an object in an external database. It has the attributes **DBName** and **Accession** and in general it is written as DBName:Accession. The DBName is the name of an external database, while Accession is the identification of the current object in the external database. For example in the DBxRef "EC:1.1.1.1" the DBName = EC (Enzyme Commission) and the Accession = 1.1.1.1.
Function: the name or description of a biological function, such as "alcohol dehydrogenase".
Feature: the nature of a subsequence, such as "mRNA", "gene", "CDS", etc..
Association: represents a relationship between 2 objects *s* (source) and *t* (target) of basic entities. We say that *s* is associated to *t* . We can classify the associations and store the type(s) of each association. Currently, we are using this entity to represent the parent relationship of annotation objects (described below).
Annotation: a specific case of association where the relationship between the two associated objects is uncertain. It is used when an annotation method tentatively assigns a basic database entity object to another basic database entity object.
An Annotation has the following attributes:

- **Method**: the annotation method that suggested the association.
- **Score**: (optional) the score of the association, as given by the annotation method. In general, the score represents the reliability of the annotation as assessed by an individual method.
- **Parent**: (optional) indicates one or more parent annotations. The parent annotations are the annotations one level higher than the current annotation. Currently, this attribute is used to describe the embedding of feature associations. For example, in the annotation of a given subsequence as being an "mRNA", the parent annotation is the annotation of the corresponding subsequence with the feature "gene".

The annotation objects are classified according to the source and target types involved in the proposed assignment. In the current version we use the following Annotation objects:

- **SubseqAnnotation**: represents the annotations of the subsequences. We have the following SubseqAnnotation objects:
  - **SubseqFeatureAnnotation**: represents one proposed assignment of a feature (e.g. gene, mRNA, CDS, etc.) to a subsequence.
  - **SubseqDBxRefAnnotation**: represents one proposed assignment of an external reference (e.g. EC, GO, etc) to a subsequence.
  - **SubseqFunctionAnnotation**: represents one proposed assignment of a function (e.g. "xanthine dehydrogenase") to a subsequence.
  - **DBxRefDBxRefAnnotation**: represents one proposed assignment of a DBxRef to another DBxRef (e.g. to assign an EC number to a KO (Kegg Ontology) number).
- **AnnotationMethod**: the method used to generate an annotation.

All basic entities can have multiple external names stored as synonyms using DBxRef objects and can have multiple simple **notes** stored as text. The notes have a note type and a text value.

3. The programs

The CycADS programs can be executed at the command lines. A program uses parameter values (or arguments) passed on the command lines and in the file config.properties to get all the information necessary to its execution. The following subsections will describe these commands and their parameters. We have established a pattern for the name of the parameters in the config.properties file. In general, a line in the config.properties follows this syntax: <program name>.[file.]<parameter specificity>[.regex][.<i>]=<value>, where:

- <program name>: is the name of the program or the word 'general' if the parameter will be used by many programs;
- file: indicates that the parameter value is related to a file specification;
- <parameter specificity>: is a name that will specify the parameter. In general it will indicate its usage;
- regex: indicates that the program expects a regular expression;
- <i>: is a number between *0* and *(2^16)-1*, used when the program expects to receive a list of arguments for the parameter <program name>.[file.]<parameter name>[.regex]. The number *i* indicates the position of <value> in this list of arguments. If the list does not have the value at the position *i-1* then the value at *the position i* will be not treated by the program and it will be not included in the list. If there is only one argument in the list to pass to the program the number *i=0* is optional;
- <value>: corresponds to the argument passed to the program or the value assigned to the parameter. The words enclosed by '[ ]' appears only in some parameters. All arguments passed by the command line can be passed by the config.properties, but the command line arguments will overwrite the arguments of the same parameters passed in the config.properties, moreover these arguments in config.properties will be used as default value in a question window to confirm the choice.

3.1. Loader programs

CycADS can import (or load) data from several sources and formats. CycADS has the following loader programs:

GBKLoader

Import data from GenBank Flat File Format (http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html). A feature annotation (CDS, mRNA, gene, etc.) in the GenBank file is described with its location (subsequence) in the sequence and other informations (e.g. name, parent, synonyms, EC number, etc.) presente as a list of tags. We will say that one *tag* is composed of a *tag name* and, in general, a *tag value*. This program can store in the database all types of entities except DBxRefDBxRefAnnotation. Beyond the entities, the program can create and store:

- synonyms for Sequences, Subsequences and SubseqFeatureAnnotations and
- •notes for Sequences and SubseqFeatureAnnotations.

The line command is
org.cycads.ui.loader.GBKLoader <fileName> <seqDBName>

where:
fileName: name and path of the Genbank file to load.
seqDBName: database name of the sequence *accession.* The sequence accession is located in the genbank file. The DBxRef object created with the *seqDBName* and *accession* will be stored in the database and will be the synonym of the sequence.
After these parameters, the loader accepts many parameters by config.properties to interpret the genbank file content and to transform it in the CycADS database format. A <type> (or *feature config type name)* in the parameter name in the config.properties means that the respective argument is applied only to features of the type <type>.

The parameters in the config file are:
GBKLoader.file.sequence.dbName: the same as the *seqDBName* line command parameter.
GBKLoader.file.fileName: the same as the *fileName* command line parameter.
GBKLoader.file.sequence.description.noteType: the note type of the note entity to store the description (or definition) of the sequence indicated in the genbank file.
GBKLoader.file.sequence.comment.noteType: the note type of the note entity to store the notes (or comments) of the sequence indicated in the genbank file.
GBKLoader.file.removeType.regex.<i>: if the argument matches with the type name of a feature in the genbank file then this feature will be thrown out and will not be loaded.
GBKLoader.file.changeType.regex.<i> and **GBKLoader.file.changeType.newValue.<i>:** if *GBKLoader.file.changeType.regex.i* argument matches with the type name of a feature in the genbank file then the type name of this feature will change to *GBKLoader.file.changeType.newValue.i*.
GBKLoader.file.changeConfigType.regex.<i> and **GBKLoader.file.changeConfigType.newValue.<i>:** these parameters are used to get the feature config type name (the word <type> in the parameter name). If *GBKLoader.file.*changeConfigType*.regex.i* argument matches with the type name of a feature in the genbank file then the feature config type name of this feature will change to *GBKLoader.file.*changeConfigType*.newValue.i.* For instance, if we set *GBKLoader.file.*changeConfigType*.regex.i=.*RNA* and *GBKLoader.file.*changeConfigType*.newValue.i=RNA* then we can group all feature types that finish with the word 'RNA' (like mRNA, ncRNA, tRNA, etc.) and treat them with the same parameters in the config.properties file.
GBKLoader.file.<type>.methodName: indicates the name of the annotation method used when creating an entity SubseqFeatureAnnotation.
GBKLoader.file.<type>.tagOperation.<i>: this parameter says the operation to execute in the tag name or tag value of a tag in the genbank file. The operation <i> is executed after the operation <i-1>. The arguments (or operations) can be: remove, split, changeTagName, changeTagValue and copy. An operation *op* is executed over a tag if the additional arguments GBKLoader.file.<type>.tagOperation.<i>.<op>.tagName.regex and GBKLoader.file.<type>.tagOperation.<i>.<op>.tagValue.regex match with, respectively, tag name and tag value of the tag. The possible operations are:

- **remove:** remove the tag of the feature tag list;
- **split:** split the tag value string around matches of the regular expression given by the additional argument GBKLoader.file.<type>.tagOperation.<i>.split.separator.regex;
- **changeTagName:** replace the tag name by the string given by GBKLoader.file.<type>.tagOperation.<i>.changeTagName.newTagName argument;
- **changeTagValue:** replace each substring of the tag value string that matches the regular expression given by the additional argument GBKLoader.file.<type>.tagOperation.<i>.changeTagValue.substSourceTagValue.regex with the replacement given by the additional argument GBKLoader.file.<type>.tagOperation.<i>.changeTagValue.substTargetTagValue. These additional arguments can be a list of arguments. In this case the operation is executed for each pair GBKLoader.file.<type>.tagOperation.<i>.changeTagValue.substSourceTagValue.regex.<j> and GBKLoader.file.<type>.tagOperation.<i>.changeTagValue.substTargetTagValue.<j>
- **copy:** create a new tag with the new tag name given by GBKLoader.file.<type>.tagOperation.<i>.copy.newTagName argument and the new tag value equal to the tag value.

GBKLoader.file.debug.outputGbkFile: the name of the file to output the updated genbank file to after the editions.
GBKLoader.file.<type>.featureSynonym.tagName.regex.<i>,GBKLoader.file.<type>.featureSynonym.tagValue.regex.<i> and **GBKLoader.file.<type>.featureSynonym.dbName.<i>:** these parameters are used to assign synonyms to the SubseqFeatureAnnotation entity created with the feature information in the genbank file. If a tag name and the tag value of this feature match with, respectively, GBKLoader.file.<type>.featureSynonym.tagName.regex.<i> and

GBKLoader.file.<type>.featureSynonym.tagValue.regex.<i> then CycADS will create a synonym to the feature with the DBName equal to the argument GBKLoader.file.<type>.featureSynonym.dbName.<i> and accession equal to the tag value.
GBKLoader.file.<type>.featureParent.tagName.regex.<i>, GBKLoader.file.<type>.featureParent.tagValue.regex.<i> and **GBKLoader.file.<type>.featureParent.dbName.<i>:** these parameters are used to assign parents to the SubseqFeatureAnnotation entity created with the feature information in the genbank file. If a tag name and the tag value of this feature match with, respectively, GBKLoader.file.<type>.featureParent.tagName.regex.<i> and GBKLoader.file.<type>.featureParent.tagValue.regex.<i> then CycADS will create the parents of the feature with the entities that have a synonym with DBName equal to GBKLoader.file.<type>.featureParent.dbName.<i> and accession equal to the tag value.
GBKLoader.file.<type>.subseqSynonym.tagName.regex.<i>,GBKLoader.file.<type>.subseqSynonym.tagValue.regex.<i> and **GBKLoader.file.<type>.subseqSynonym.dbName.<i>:** these parameters are used to assign synonyms to the Subsequence entity of the SubseqFeatureAnnotation entity. The parameters usage are similar to the SubseqFeatureAnnotation synonym creation.
GBKLoader.file.<type>.subseqDbxRefAnnotation.tagName.regex.<i>,GBKLoader.file.<type>.subseqDbxRefAnnotation.tagValue.regex.<i>,GBKLoader.file.<type>.subseqDbxRefAnnotation.dbName.<i> and **GBKLoader.file.<type>.subseqDbxRefAnnotation.methodName.<i>:** these parameters are used to create SubseqDbxRefAnnotation objects with the annotation method equal to GBKLoader.file.<type>.subseqDbxRefAnnotation.methodName.<i>. They are used with the feature information in the genbank file such that the subsequence is the subsequence of the SubseqFeatureAnnotation entity created, the DBxRef is GBKLoader.file.<type>.subseqDbxRefAnnotation.dbName.<i> : <tag value> and the tag name and its tag value match with, respectively, GBKLoader.file.<type>.subseqDbxRefAnnotation.tagName.regex.<i>, and GBKLoader.file.<type>.subseqDbxRefAnnotation.methodName.<i>.
GBKLoader.file.<type>.subseqFunctionAnnotation.tagName.regex.<i>,GBKLoader.file.<type>.subseqFunctionAnnotation.tagValue.regex.<i> and **GBKLoader.file.<type>.subseqFunctionAnnotation.methodName.<i>:** these parameters are used to create SubseqFunctionAnnotation objects with the annotation method equal to GBKLoader.file.<type>.subseqFunctionAnnotation.methodName.<i>. They are used with the feature information in the genbank file such that the subsequence is the subsequence of the SubseqFeatureAnnotation entity created, the Function name is the tag value, and the tag name and its tag value match with, respectively, GBKLoader.file.<type>.subseqDbxRefAnnotation.tagName.regex.<i> and GBKLoader.file.<type>.subseqDbxRefAnnotation.methodName.<i>.

GFF3Loader

Import data from GFF3 File Format (http://www.sequenceontology.org/gff3.shtml). The GFF3 file is used to describe sequence feature annotations (CDS, mRNA, gene, etc.). This program loads CDS, RNA and gene feature annotations (SubseqFeatureAnnotation entities). The exons are considered just to create the RNA introns and, consequently, the CDS introns. The others features types are ignored by this program.
Each feature annotation in the GFF3 file is described in a line of 9 tab-delimited columns. There are columns to inform: feature annotation type, feature annotation location (source sequence synonym, start, end and strand), annotation method (or source) and annotation score. Behind these well defined columns there is one column to put generic informations (or attributes) about the feature annotation using a list in the format "tag name=tag value". We will call this list of feature attribute list. This program can store in the database all types of entities except DBxRefDBxRefAnnotation and SubseqDBxRefAnnotation. Behind the entities, the program can create and store:
synonyms for Sequences, Subsequences and SubseqFeatureAnnotations and
notes for SubseqFeatureAnnotations.
The line command syntax is
org.cycads.ui.loader.GFF3Loader <fileName> <organismNumber> <organismName> <seqDBName> <seqVersion>
where:
fileName: name and path of the GFF3 file to load. ▪ **organismNumber**: NCBI taxon ID of the organism to be loaded.
organismName: name of the organism to be used when creating new organisms.
seqDBName: database name of the sequence *accession.* The sequence accession is in the first column of the GFF3 file. One DBxRef object is created with the *seqDBName* and *accession* to be a synonym of the sequence.
seqVersion: the sequence version to be used when creating new sequences.
After these parameters, the loader accepts many parameters by config.properties to interpret the GFF3 file content and to transform it in the CycADS database format. The parameter names in the config file follow this syntax:

GFF3Loader.file[.attribute][.<type>][.<source>].<parameter specificity>[.regex][.<i>]. Moreover the name parts described in the beginning of this section we have:
<type>: (or *feature config type name)* means that the respective argument is applied only to features of the type <type>;
<source>: means that the respective argument is applied only to features where source (the second column in the GFF3 file) is equal to <source>;
attribute: is used if the argument is relative to the list of feature attributes in the GFF3 file.

The parameters in the config.properties are:
GFF3Loader.loader.fileName: the same as the *fileName* line command parameter.
GFF3Loader.loader.organismNumber: the same as the organismNumber line command parameter.
GFF3Loader.loader.organismName: the same as the organismName line command parameter.
GFF3Loader.loader.sequenceDbName: the same as the seqDBName line command parameter.
GFF3Loader.loader.sequenceVersion: the same as the seqVersion line command parameter.
GFF3Loader.file.geneConfigType, GFF3Loader.file.exonConfigType, GFF3Loader.file.rnaConfigType and **GFF3Loader.file.cdsConfigType**: for each feature annotation type (gene, RNA, CDS and exon) indicate the type name to use in the parameter names in the config.properties. This type name will be stored also in the database as a feature object.
GFF3Loader.file[.<type>].type.regex: if GFF3Loader.file[.<type>].type.regex matches with the type name of the feature annotation to be load then the feature name of the SubseqFeatureAnnotation object will be <type>.
GFF3Loader.file[.<type>][.<source>].methodName: gives the annotation method name for the annotations of the type=<type> and source=<source>.
GFF3Loader.file.attribute[.<type>][.<source>].removeTag.regex[.<i>] if the tag name of a attribute of the feature matches with GFF3Loader.file.attribute[.<type>][.<source>].removeTag.regex[.<i>] then this attribute will not be load by the program.
GFF3Loader.file.attribute[.<type>][.<source>].replaceTagName.regex[.<i>] and **GFF3Loader.file.attribute[.<type>][.<source>].replaceNewTagName.regex[.<i>]** if the tag name of a attribute of the feature matches with GFF3Loader.file.attribute[.<type>][.<source>].replaceTagName.regex[.<i>] then the tag name of this attribute will be changed to GFF3Loader.file.attribute[.<type>][.<source>].replaceNewTagName.regex[.<i>]
GFF3Loader.file.attribute[.<type>][.<source>].annotationCompletSynonymTag.regex and **GFF3Loader.file.attribute[.<type>][.<source>].annotationCompletSynonymSeparator** if the tag name of a attribute of the feature matches with GFF3Loader.file.attribute[.<type>][.<source>].annotationCompletSynonymTag.regex then the tag value of this attribute will be used to create a DBxRef as synonym of the SubseqFeatureAnnotation object. The DBName and Accession of the DBxRef are in the tag value separated by GFF3Loader.file.attribute[.<type>][.<source>].annotationCompletSynonymSeparator
GFF3Loader.file.attribute[.<type>][.<source>].annotationSynonymAccessionTag.regex and **GFF3Loader.file.attribute[.<type>][.<source>].annotationSynonymDbName** if the tag name of a attribute of the feature matches with GFF3Loader.file.attribute[.<type>][.<source>].annotationSynonymAccessionTag then the tag value of this attribute will be used as accession of a DBxRef to assign a synonym to the SubseqFeatureAnnotation object. The DBName of the DBxRef will be the argument GFF3Loader.file.attribute[.<type>][.<source>].annotationSynonymDbName
GFF3Loader.file.attribute[.<type>][.<source>].subsequenceSynonymAccessionTag.regex and **GFF3Loader.file.attribute[.<type>][.<source>].subsequenceSynonymDbName** if the tag name of a attribute of the feature matches with GFF3Loader.file.attribute[.<type>][.<source>].subsequenceSynonymAccessionTag then the tag value of this attribute will be used as accession of a DBxRef to assign a synonym to the Subsequence object. The DBName of the DBxRef will be the argument GFF3Loader.file.attribute[.<type>][.<source>].subsequenceSynonymDbName
GFF3Loader.file.attribute[.<type>][.<source>].parentAccessionTag.regex and **GFF3Loader.file.attribute[.<type>][.<source>].parentDbName[.<i>]** if the tag name of a attribute of the feature matches with GFF3Loader.file.attribute[.<type>][.<source>].parentAccessionTag.regex then the tag value of this attribute will be used as accession of a DBxRef to try to find an annotation object to be the parent of the SubseqFeatureAnnotation object. The DBName of the DBxRef will be the argument GFF3Loader.file.attribute[.<type>][.<source>].parentDbName[.<i>] and the DBxRef must be a synonym of the parent annotation object.
GFF3Loader.file.attribute[.<type>][.<source>].noteTypeTag.regex and **GFF3Loader.file.attribute[.<type>][.<source>].noteTypeValue[.<i>]** if the tag name of a attribute of the feature matches with GFF3Loader.file.attribute[.<type>][.<source>].noteTypeTag.regex[.<i>] then a note will be add to the SubseqFeatureAnnotation object. The note type will be GFF3Loader.file.attribute[.<type>][.<source>].noteTypeValue[.<i>] and the note value will be the tag value.
GFF3Loader.file.attribute[.<type>][.<source>].functionTag.regex and **GFF3Loader.file.attribute[.<type>][.<source>].functionMethodName** if the tag name of a attribute of the feature

matches with GFF3Loader.file.attribute[.<type>][.<source>].functionTag.regex[.<i>] then a SubseqFunctionAnnotation will be created with the function name equals to the tag value and the annotation method name equals to GFF3Loader.file.attribute[.<type>][.<source>].functionMethodName[.<i>]

KOLoader

This program loads data from a specific file from KEGG (ftp://ftp.genome.jp/pub/kegg/genes/ko).
This program will create a DBxRef to represent each KO record in the file. This KO DBxRef object has the dbname equals to "KO" and the accession equals to the first six characters in the entry field of the KO record. If there is an EC number in the definition field of the KO record then the program will create a DBxRefDBxRefAnnotation object to represent this link KO to EC. Other DBxRefDBxRefAnnotation object will be created for each external reference objects in the DBLinks field of the KO record. The text in the definition field, except the EC numbers references, will be stored as the definition note of the KO DBxRef object. The line command is
org.cycads.ui.loader.KOLoader <fileName>
where fileName is the name and path of the KO file to load.
The parameters in the config.properties are:
KOLoader.fileName: the same as the *fileName* line command parameter.
KOLoader.ECAnnotationMethodName: the annotation method name of the DBxRefDBxRefAnnotation object created to represent the link KO to EC numbers.
KOLoader.DBLinkMethodName: the annotation method name of the DBxRefDBxRefAnnotation object created to represent the link with others external reference objects (except EC).
KOLoader.file.DBLink.dbName.regex[.<i>] and **KOLoader.file.DBLink.newDbName[.<i>]**: these parameters are used to change the DBName of a DBxRef created to represent a external reference object in the DBLink field of the KO record in the KO file. If KOLoader.file.DBLink.dbName.regex[.<i>] matches with a DBName in the DBLink field of the KO record in the file then the DBName to be stored will be KOLoader.file.DBLink.newDbName[.<i>]. There are others parameters in the config.properties to configure the file format and must be changed only if the KO file format is changed. In the case of the KO file format is changed, please try to get the newest version of CycADS or contact the support.

The following loaders programs are programs to create annotations to DBxRefs or to create synonyms. The program load text files with columns delimited by a string (e.g. *tab* caracter). It is mandatory to specify two columns: source column and DBxRef column. The source column must have the DBxRefs to identify the source entities. The source entity type change accordingly the loader program. For the programs that create annotation objects we can have more two columns: score column and method column.

SubseqDbxrefAnnotationLoader

This program loads SubseqDbxrefAnnotation objects created from a text file with columns delimited by a string. The column delimiter string is given by one parameter in the config.properties file. The file must have at least 2 columns: the subsequence identifications column (or subsequence DBxRefs column) and the DBxRef identifications column. Given a line in the file, for each pair (Subsequence, DBxRef) identified one SubseqDbxrefAnnotation object is created. The subsequence identification (or subsequence DBxRef) is a DBxRef which must be a subsequence synonym or an association synonym with the association source being a subsequence. If the subsequence DBxRef don't reach a subsequence (by the subsequence synonym or association synonym) then the program will do nothing. The line command is **org.cycads.ui.loader.SubseqDbxrefAnnotationLoader <fileName> <organismNumber> <methodName> <sourceColumnIndex> <sourceDBName> <targetColumnIndex> <targetDBName> <scoreColumnIndex> <methodColumnIndex>** where
fileName: name and path of the file to load.
methodName: the default method name to be used to create the SubseqDbxrefAnnotation object. This argument will be used only if there isn't one column to give the method name or the content of this column is blank or empty.
sourceColumnIndex: the number of the subsequences identifications column.
sourceDBName: the external database name of the subsequences identifications. This database name will be used if the subsequences identifications doesn't have a database name.
targetColumnIndex: the number of the DBxRef column.
targetDBName: the external database name of the DbxRef. This database name will be used if the DBxRef identification doesn't have a database name.
scoreColumnIndex: the number of the score column. If there is not a value in the column indicate by this parameter then the annotation object will not have a score.

methodColumnIndex: the number of the method column. If there is not a value in the column indicate by this parameter then the method name of the annotation object will be the method name default indicated in the methodName parameter.

The parameters in the config.properties are: The same as above, which will be used when they are missing in the command line parameters. Note that confirmation is asked to the user with an input dialogbox. Additional parameters about the file format can/must be set such as column separator, line comments chars and so on..
subseqDbxrefAnnotationLoader.organismNumber: the same as the organismNumber line command parameter.
subseqDbxrefAnnotationLoader.fileName: the same as the fileName command line parameter.
subseqDbxrefAnnotationLoader.methodName: the same as the methodName command line parameter.
subseqDbxrefAnnotationLoader.subseqColumnIndex: the number of the subsequences identification column.
subseqDbxrefAnnotationLoader.subseqDBName: the subsequence DB name.
subseqDbxrefAnnotationLoader.dbxrefColumnIndex: the number of the DBxRef column.
subseqDbxrefAnnotationLoader.dbxrefDBName: the DBxRef external DB name.
subseqDbxrefAnnotationLoader.scoreColumnIndex: the number of the score column.
subseqDbxrefAnnotationLoader.methodColumnIndex: the number of the score method column.
subseqDbxrefAnnotationLoader.assocTypeName: the association type (Function assignment)
subseqDbxrefAnnotationLoader.file.sourcesDelimiter: the delimiter string for the sources identifications.
subseqDbxrefAnnotationLoader.file.targetsDelimiter: the delimiter string for the targets identifications.
subseqDbxrefAnnotationLoader.file.sourceColumnDelimiter: the delimiter string for the sourceColumn.
subseqDbxrefAnnotationLoader.file.targetColumnDelimiter: the delimiter string for the sourceColumn.
subseqDbxrefAnnotationLoader.file.scoreDelimiter: the delimiter string for the score.
subseqDbxrefAnnotationLoader.file.methodDelimiter: the delimiter string for the method name.
subseqDbxrefAnnotationLoader.file.lineComment: the string which indicate a comment line.
subseqDbxrefAnnotationLoader.file.columnSeparator: the string of the column separator.
subseqDbxrefAnnotationLoader.file.sourcesSeparator: the separator string of the sources identifications.
subseqDbxrefAnnotationLoader.file.targetsSeparator: the separator string of the targets identifications.
subseqDbxrefAnnotationLoader.file.removeLineRegex: the regular expression defining the pattern for the matching lines to remove.

EntityDbxrefAnnotationLoader

This program loads EntityDbxrefAnnotation objects created from a text file with columns delimited by a string. The column delimiter string is given by one parameter in the config.properties file. The file must have at least 2 columns: the entity identifications column (or entity DBxRefs column) and the DBxRef identifications column. Given a line in the file, for each pair (Entity, DBxRef) identified, one EntityDbxrefAnnotation object is created. The entity identification (or entity DBxRef) is a DBxRef which must be an entity synonym. If the entity DBxRef don't reach an entity (by the entity synonym) then the program will do nothing.
The line command is **org.cycads.ui.loader.EntityDbxrefAnnotationLoader <fileName> <methodName> <sourceColumnIndex> <sourceDBName> <targetColumnIndex> <targetDBName> <scoreColumnIndex>** <*optional* **methodColumnIndex>**
where
fileName: name and path of the file to load.
methodName: the default method name to be used to create the EntityDbxrefAnnotation object.
sourceColumnIndex: the number of the entity identifications column.
sourceDBName: the external database name of the entity identifications. This database name will be used if the entity identification doesn't have a database name.
targetColumnIndex: the number of the DBxRef column.
targetDBName: the external database name of the DbxRef. This database name will be used if the DBxRef identification doesn't have a database
scoreColumnIndex: the number of the score column. If there isn't a value in the column indicate by this parameter then the annotation object will not have a score.
methodColumnIndex: the number of the method column. If there is not a value in the column indicate by this parameter then the method name of the annotation object will be the method name default indicated in the methodName parameter.

The parameters in the config.properties are:

The same as above, which will be used when they are missing in the command line parameters. Note that confirmation is asked to the user with an input dialogbox. Additional parameters about the file format can/must be set such as column separator, line comments chars and so on.. (as shown in the previous SubseqDbxrefAnnotationLoader loader).

DbxrefDbxrefAnnotationLoader

This program loads DbxrefDbxrefAnnotation objects created from a text file with columns delimited by a string. The column delimiter string is given by one parameter in the config.properties file. The file must have at least 2 columns: the dbxrefSource identifications column (or dbxrefSource DBxRefs column) and the dbxrefTarget identifications column. Given a line in the file, for each pair (dbxrefSource, dbxrefTarget) identified one DbxrefDbxrefAnnotation object is created. The dbxrefSource and the dbxrefTarget identifications are DBxRefs. They will be created if not exist. The line command is **org.cycads.ui.loader.DbxrefDbxrefAnnotationLoader <fileName> <methodName> <sourceColumnIndex> <sourceDBName> <targetColumnIndex> <targetDBName> <scoreColumnIndex> <***optional* **methodColumnIndex>**
where
fileName: name and path of the file to load.
methodName: the default method name to be used to create the DbxrefDbxrefAnnotation object.
sourceColumnIndex: the number of the entity identifications column.
sourceDBName: the external database name of the entity identifications. This database name will be used if the subsequence identification doesn't have a database name.
targetColumnIndex: the number of the DBxRef column.
targetDBName: the external database name of the DbxRef. This database name will be used if the DBxRef identification doesn't have a database
scoreColumnIndex: the number of the score column. If there is not a value in the column indicate by this parameter then the annotation object will not have a score.
methodColumnIndex: the number of the method column. If there is not a value in the column indicate by this parameter then the method name of the annotation object will be the method name default indicated in the methodName parameter.
The parameters in the config.properties are: The same as above, which will be used when they are missing in the command line parameters. Note that confirmation is asked to the user with an input dialogbox. Additional parameters about the file format can/must be set such as column separator, line comments chars and so on..

EntitySynonymLoader

This program loads EntitySynonym objects created from a text file with columns delimited by a string. The column delimiter string is given by one parameter in the config.properties file. The file must have at least 2 columns: the Entity identifications column (or Entity DBxRefs column) and the Synonym identifications column. Given a line in the file, for each pair (Entity, Synonym) identified one EntitySynonym object is created. The Entity identification is a DBxRef which must be an entity synonym. If the Entity DBxRef don't reach an entity synonym then the program will do nothing. The line command is **org.cycads.ui.loader.EntitySynonymLoader <fileName> <sourceColumnIndex> <sourceDBName> <synonymColumnIndex> <synonymDBName>**
where
fileName: name and path of the file to load.
sourceColumnIndex: the number of the entity identifications column.
sourceDBName: the external database name of the entity identifications. This database name will be used if the entity identification doesn't have a database name.
synonymColumnIndex: the number of the Synonym column.
synonymDBName: the external database name of the Synonym. This database name will be used if the Synonym identification doesn't have a database These parameters can be set in the config.properties.

DbxrefSynonymLoader

This program loads DbxrefSynonym objects created from a text file with columns delimited by a string. The column delimiter string is given by one parameter in the config.properties file. The file must have at least 2 columns: the Dbxref identifications column and the Synonym identifications column. Given a line in the file, for each pair (Dbxref, Synonym) identified one DbxrefSynonym object is created. The Dbxref and Synonym identifications are DBxRef. If the DBxRef don't exist then the program will create one. The line command is **org.cycads.ui.loader.DbxrefSynonymLoader <fileName> <sourceColumnIndex> <sourceDBName> <synonymColumnIndex> <synonymDBName>**

where
fileName: name and path of the file to load.
sourceColumnIndex: the number of the dbxref identifications column.
sourceDBName: the external database name of the dbxref identifications. This database name will be used if the dbxref identification doesn't have a database name.
synonymColumnIndex: the number of the Synonym column.
synonymDBName: the external database name of the Synonym. This database name will be used if the Synonym identification doesn't have a database These parameters can be set in the config.properties.

3.2. Program to extract annotations to a file

CycADS can export data in the Pathologic file format used by Biocyc databases (Pathway Tools). Data output can be filtered using the command line parameters and mostly using the configuration file (config.properties).

The command line

CycADS has the following extractor programs: **org.cycads.ui.extract.cyc.AnnotationGenerator <fileName|folderName> <organismNumber> <seqDbname:seqAccession> <seqVersion> <sequenceLocation> <ecThreshold> <goThreshold> <fileFormat>**
where
fileName|folderName: name and path of the file to generate or path to the folder when multiple PF files options is used (see fileFormat parameter bellow).
organismNumber: NCBI taxon ID of the organism which data will be extracted.
seqDbname:seqAccession: the sequence identification (database name and accession of the sequence, ':' separated). '*' char will select all the sequences.
seqVersion: a valid sequence version or all the versions of the sequence using the '*' char.
sequenceLocation: 'y' or 'n' as boolean char to export respectively sequence location or not.
ecThreshold: annotation methods evidences for EC (Enzyme Commission numbers) are grouped in an "EC annotation cluster" when they are loaded. So that EC cluster got a cumulative score for EC. EC with a cluster score bellow the ecThreshold will not be extracted.
goThreshold: the same as ECthreshold but for the GO. 3.2.2)**.**
fileFormat: 1 = pf file ; 2 = annotationByLine (each line 1 GO, EC, KO, etc. annotation) ; 3 = functionByLine (each line 1 CDS, TRNA, etc function) ; 4 = Multiple PF Files (for example one per sequence returned by the '*' char set for seqDbname:seqAccession parameter).

Config.properties

CycADS can filter further using the config properties where many parameters can be set to produce a rich output for BioCyc databases. Most of these parameters accept regular expressions and filters for the best flexibility. Parameters names follow the same expression using defined types to simplify the properties writing. The program call an "object getter" (annotation cluster) for each parameter. This object getter can extract annotation data through the location (loc) indicated, make changes or get another filtered object.
Parameters names definition
AnnotationGenerator.<fileformat>.<parameter name>[.<type>][.<method>][.loc][.regex][.<i>] where
<fileformat>: indicates that the parameter value is related to a file specification;
pf: apply to PFFile specification
<type>: (or *feature config type name)* means that the respective argument is applied only to features of the type <type>;
<parameter name>: means that the respective argument is applied only to object;
fileName: the same as the command line first
organismNumber: the taxon id of the organism.
sequenceSynonym: a sequence synonym in the database (or '.' for all sequences).
sequenceVersion: the sequence version to be extracted
feature: the name of the considered features (ie "CDS", "*.RNA", ..)
productType: product type and changes operations on it which can be made in accordance with the target output format (location, replace.regex, replace.replacement).

geneName: the accessions of the genes (location of gene data).
geneSynonyms: the accessions of the gene synonyms (location of gene synonyms data).
geneAnnot: the location of the gene annotations (location of gene annotations).
geneComments : the comments or output messages associated with the genes.
ko: the accessions of the KOs
koMsg: the output message associated with the KOs.
functionGOs:
goMsg: the output message associated with the GOs.
functionECs:
ecMsg: the output message associated with the ECs.
dblinks and geneDblinks: identification of data having to appear as dblink.
score: apply to ko, functionGOs and functionECs. A default score ((method.valueDefault) and weighting (method.weight) for each method defined in method.regex can be set. This is useful for filtering the output.
threshold: a double value indicating the threshold for ec and go respectively. Only ec or go annotations having a score higher than the value will be extracted.
scoreFilterByAnnotationType and methodFilterByAnnotationType: respectively score and method can be filtered by type Function assignment or DbxRef assignment.

outFormat: an integer which indicate the output format,
1: pf file
2: annotationByLine (each line 1 GO, EC, KO, etc. annotation)
3: functionByLine (each line 1 CDS, TRNA, etc function).
idNoteType: the note id type ("CycID": uses CycADS identification)
methodSeparator: the separator string for the methods.
methodPathSeparator: the separator string for the methods paths.
annotationScoreDefault: double value for the default annoation score.
pf.header: the header of the generated PF file
pf.functionName: function name or accession for the FUNCTION attribute in PF Files.
pf.functionName.loc: where functionName are extracted from CycADS.
pf.functionSynonyms.loc: where function synonyms are extracted from CycADS.
pf.functionComments.loc: where function comments are extracted from CycADS.
pf.functionSSequence.loc: where function subsequences are extracted from CycADS.
pf.AnnotationGenerator.pf.sequenceLocation: exports the sequence location to PF Files or not (y/n).
pf.AnnotationGenerator.pf.geneComment.separator: the separator of genes comments for the PF File output.
pf.AnnotationGenerator.pf.functionComment.separator: the separator of functions comments for the PF File output.

<loc>: the field location (where the object getter can find the data)
<type>: the scope of the parameter regarding the feature (CDS|TRNA|MISCRNA);
<method>: the scope of the parameter regarding the method;
regex: used when the program expect a regular expression;
<i>: an integer value between *0* and *(2^16)-1* when the program expect a list of arguments.

Parameters values (They can be double, integer or string)
Regular expressions are built using :
locType=P (Parent)|FA (FunctionAnnotation)| XA (dbxrefAnnotation)| SY (Synonym)| NO (Note)| SE (Sequence)| SS (Subsequence)| F (Function)| V (NoteValue or Dbxref accession or Function Name)| XV (Dbxref of dbxrefannotation)| XR.<name> (set of dbxrefs)
Other expression types are defined to use in the syntax of the parameters values and get objects in accordance with the program state machine :
AT: Annotations where Entity is the target
AS: Annotations where Entity is the source
SO: Source
TA: Target
AY: Annotation Types
ME: Method

SC: Score
EN: End
BE: Begin
NA: Name
OR: Organism
SE: Sequence
VE: Version
SS: Subsequence
TY: Entity Type
SY: Synonym
NO: Note
NT: Note Type
NV: Note Value
DB: Database Name - DbxRef
AC: Accession - DbxRef
ST: to String - All
SI: Size of Sequence
the '.' char ask to the program to get a data object using the expression
filter expressions (#<*comparison operator*><*string value*>#)
comparison operator: ==|<=|>=|<<|>>|=>|=<|=!
string value: a defined type or another regular expression

3.3. Auxiliary programs

org.cycads.ui.tools.CleanColumn <fileName> <cleanColumnIndex> <columnSeparator> <lineComment> <cleanExpression> <outputFileName>
This tool can clean the indicated column using the ***cleanExpression*** regular expression.

org.cycads.ui.tools.RemoveLineStringColumn <fileName> <removeColumnIndex> <columnSeparator> <lineComment> <removeExpression> <outputFileName>
This tool can remove the lines using the ***removeExpression*** regular expression matching in the indicated column.

org.cycads.ui.tools.RetrieveStringColumn <fileName> <retrieveColumnIndex> <columnSeparator> <lineComment> <retrieveExpression> <outputFileName>
This tool can retrieve the lines using the ***retrieveExpression*** regular expression matching in the indicated column.

org.cycads.ui.tools.SplitFastaFile <**fileName**> <**searchColumnIndex**> <**columnSeparator**> <**searchExpression**> <**outfileExtension**>
This tool can split a multifasta file using a regular expression matching in the indicated column of fasta headers strings. It uses the regular expression matches for the outfiles names.

4. Usage examples

As show in ***T. castaneum*** **config.properties** example file provided in the package.
AnnotationGenerator.pf.geneName.loc.0=.PA.PA.SY(.db(#=TCOGS2#)).AC
This gets the accession of the gene synonyms (grandparents synonyms of the features in the TCOGS2 database).
AnnotationGenerator.pf.geneName.loc.1=.PA(.TA.ST(#=(?i)Gene#)).SY(.db(#=TCOGS2#)).AC
This gets the accession of the gene synonyms (parents synonyms of the features in the TCOGS2 database, where feature have a "Gene" like parent feature).

5. System requirements

Java Runtime 1.6
SQL database Management system access

External libraries: biojava, bytecode

6. Installation

Download and install Java Runtime 1.6 if not existing in your system (http://java.sun.com/javase/downloads/index.jsp).
Download and install one SGBD, for example MySQL (http://www.mysql.com/downloads/mysql/). Note that the use of MySQL is not mandatory, other SGBDs can be prefered (you have to define the JDBC driver and the connection string in the config.properties) although we didn't have done any tests on other systems. Because annotation files can include large amino acids or nucleotides sequences data, you will probably have to set your SGBD specifications depending on the client or server SGBD used and the size of the data you want to collect. For MySQL it is necessary to increase the value of the max_allowed_packet variable (http://dev.mysql.com/doc/refman/ 5.0/en/packet-too-large.html), at least 16M or more.
Download and install the corresponding JDBC driver (Connector/J/5.1.X for the MySQL choice, http://www.mysql.com/downloads/ connector/j/5.1.html).

Create an empty database and a user with grant privileges on it. Then execute the SQL script cycads1.3.sql provided in the software package to build the CycADS database schema. SQL instructions are added for automatic triggers at the end of the script. They are optional with the current release 1.3 of CycADS.

Download CycADS package at http://pbil.univ-lyon1.fr/software/ cycads/cycads.zip

Download CycADS external libraries at http://pbil.univ-lyon1.fr/ software/cycads/libraries.zip

Unzip the CycADS software package in a directory of your choice and the libraries archive in the same directory.

The connection parameters to the CycADS DB have to be set in the **config.properties** file :
general.sql.driverName=*yourJDBCdrivername* (**com.mysql.jdbc.Driver** for MySQL)
general.sql.connectionUrl=jdbc:*yourSGBD*://*server*:*port*/*yourdatabasename (ie **jdbc:mysql://localhost:3306/cycads)***
general.sql.usr=*yourusername* general.sql.pass=*youruserpass*

Set up the config properties file in accordance with the data you want to load and/or extract.

Launch the CycADS AnnotationCollector program or (later) the AnnotationExtractor you need (respectively loaders and extractors described previously), using the command line :
$ java [Options] cycads1.3.jar:biojava.jar:bytecode.jar:mysql-connectorjava-5.1.6-bin <programName> <programParameters> with :
Options: Java options, commonly **-Xmx512M -cp**.
Memory consumption has generally to be set between 512M and 2048M, depending on the data size.
See also http://java.sun.com/javase/6/docs/technotes/tools/ windows/java.html.
programName: one Loader or Extractor as described in section 3) of this manual.
programParameters: the parameters as in the corresponding program description or by modifying the script: cycads1.3.sh, to launch most of the loaders and extractors with provided data examples (http://pbil.univ-lyon1.fr/software/ cycads/example_tricacyc.zip)

7. Troubleshooting

Java will generate error numbers and descriptions to the standard output. The error messages can help to solve the problems.

8. Future works

Graphic user interface/control of the config.properties parameters.
Delete records or undo loads.
Config operations standard for gff3 and gbk loaders.
PFFileGenerator don't follow differents structural annotations.

Plugin or connector for metabolic networks software (such as MetExplore).